Adaptive Scheduling of Data Paths using Uppaal Tiga[*]

Israa AlAttili   Fred Houben   Georgeta Igna   Steffen Michels   Feng Zhu
Frits Vaandrager

Institute for Computing and Information Sciences
Radboud University Nijmegen
P.O. Box 9010, 6500 GL Nijmegen, The Netherlands

We apply Uppaal Tiga to automatically compute adaptive scheduling strategies for an industrial case study dealing with a state-of-the-art image processing pipeline of a printer. As far as we know, this is the first application of timed automata technology to an industrial scheduling problem with uncertainty in job arrivals.

1   Introduction

Scheduling concerns the allocation of resources to activities over time in order to achieve some goals. Scheduling problems occur in many different domains and a vast amount of research has been carried out in this area. However, in the research literature scheduling is usually seen as a function of known, perfect inputs: the set of jobs, their arrival times, the capacities of machines, the duration of activities, and other characteristics of the problem are assumed to be known and static [9]. It has been observed that scheduling processes in practice are driven by uncertainty [13, 15]. This uncertainty may arise due to various sources (machine breakdown, unexpected arrival of new jobs, modification of existing jobs, uncertainty of task durations,..). McKay et al. [14] even claim that the dynamic characteristics of real-world scheduling environments render the bulk of existing solution approaches for the job shop problem unusable when applied to practical problems. The number of scientific publications devoted to scheduling in a setting with uncertainty is relatively small and most of them have the flavor of AI planning rather than Operations Research [9, 1]. Altogether the problem of computing *optimal* scheduling strategies in practical settings with uncertainty is largely open. The present paper aims to address this problem.

Within the European AMETIST project [3], steps have been taken in the development of a general theory of scheduling inspired by the methodology of model checking and based on the timed automaton model of Alur and Dill [2]. In the AMETIST approach, components of a system are modeled as *dynamical systems* with a state space and a well-defined dynamics. All that can happen is expressed in terms of *behaviors* that can be generated by the dynamical systems; these constitute the semantics of the problem. Verification, optimization, synthesis and other design activities explore and modify system structure so that the resulting behaviors are correct, optimal, etc. Using this approach, the project was able to derive schedules that were of comparable quality as those that were provided by an industrial tool [4]. Most of the work within AMETIST did not address scheduling under uncertainty, with the notable exception of [1], which studies uncertainty in task durations. Recently, however, an extension of the timed automata model checker Uppaal [6] has been proposed, called Uppaal Tiga, that implements the first efficient on-the-fly algorithm for solving games based on timed game automata with respect to reachability and safety properties [5, 7]. Although still a prototype, we believe that this extension has

potential as a tool for solving scheduling problems that involve uncertainty (using the AMETIST approach). In Uppaal Tiga, systems are specified through a network of timed game automata [12]. These are timed automata in the sense of [2] where edges are marked as either controllable or uncontrollable. This defines a two player game with on the one side the *controller* (mastering the controllable edges) and on the other side the *environment* (mastering the uncontrollable edges). Winning conditions of the game are specified through TCTL formulas and for instance state that, irrespective of the strategy used by the environment player, the system player can always reach (or always avoid) certain states. In a scheduling context, uncertainty can be modeled using uncontrollable edges. Uppaal Tiga is then able to synthesize strategies for controlling the system such that scheduling objectives are met irrespective of the timing of uncontrollable edges.

In order to demonstrate the practical usefulness of Uppaal Tiga for solving scheduling problems with uncertainty, we have applied the tool to an industrial case study from Océ Technologies that concerns the scheduling of a state-of-the-art image processing pipeline of a printer. In [10], an initial version of this scheduling problem has been described and analyzed using three different modeling frameworks: timed automata (Uppaal), colored Petri nets and synchronous dataflow. None of the models in [10] incorporated uncertainty and in particular it was assumed that the arrival times of new jobs are known in advance. In reality, of course, the arrival time of new printer jobs is typically unknown (arrival times are the most significant source of uncertainty in this application domain). In this paper we use Uppaal Tiga to generate optimal scheduling strategies for scenarios in which the arrival times of certain jobs is unknown. Previous industrial applications of Uppaal Tiga deal with the synthesis of controllers [11, 8]. The present paper describes the first application to an industrial scheduling problem.

The rest of this paper is organized as follows. Section 2 recalls the printer case study and the Uppaal model from [10]. Section 3 describes how uncertainty of arrival times can be modeled and analyzed using Uppaal Tiga. Section 4 presents an overview of the results that we obtained using Uppaal Tiga and discusses how these results can be used to improve real printers/copiers. Finally, Section 5 gives some conclusions and directions for further work. We assume that the reader has some knowledge of timed automata and Uppaal (see [6] for a tutorial). The Uppaal Tiga models described in this paper are available on-line at the URL `http://www.mbsd.cs.ru.nl/publications/papers/fvaan/TigaOce/`.

2   Case Study

In this section, we introduce the Océ case study and the Uppaal model for it. The material in this section, which has been taken from [10], will be the starting point for the application of Uppaal Tiga, which we will describe in the subsequent sections of this paper. The main challenge in the case study is to compute efficient schedules for printer/copier jobs that minimize latency and maximize throughput.

2.1   Architecture

Océ systems perform a variety of image processing functions on digital documents in addition to scanning, copying and printing. Apart from local use for scanning and copying, users can also remotely use the system for image processing and printing. A generic architecture of the system studied in this paper is shown in Fig. 1. The system has two ports for input: Scanner and Controller. Users locally come to the system to submit jobs at the Scanner and remote jobs enter the system via the Controller. These jobs use the image processing (IP) components (ScanIP, IP1, IP2, PrintIP), and system resources such as memory and a USB bus for executing the jobs. Finally, there are two places where the jobs leave the
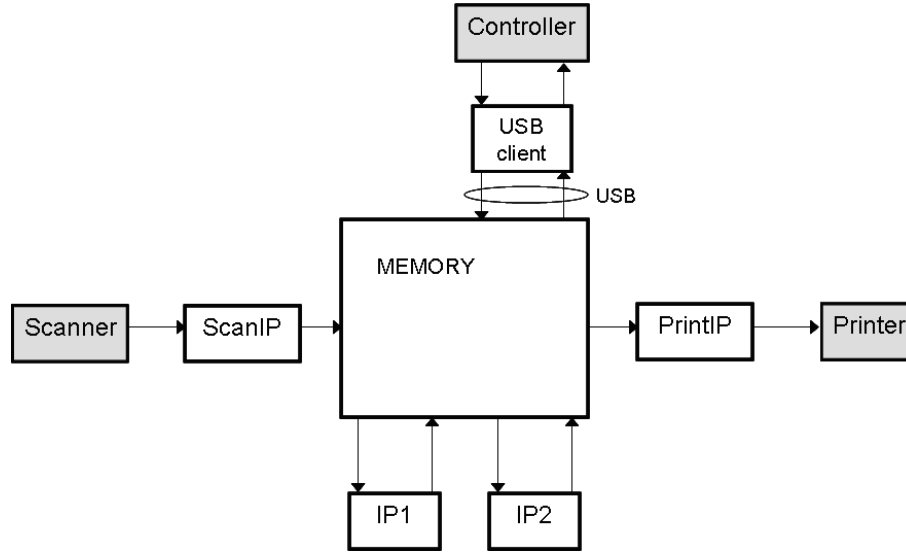
Figure 1: Architecture of Océ system

system: Printer and Controller.

The IP components can be used in different combinations depending on how a document is requested to be processed by the user. Hence this gives rise to different use cases of the system, that is, each job may use the system in a different way. The list of components used by a job defines the *datapath* for that job. Some examples of datapaths are:

- **DirectCopy:** Scanner $\rightsquigarrow$ ScanIP $\rightsquigarrow$ IP1 $\rightsquigarrow$ IP2 $\rightsquigarrow$ USBClient, PrintIP
- **ScanToStore:** Scanner $\rightsquigarrow$ ScanIP $\rightsquigarrow$ IP1 $\rightsquigarrow$ USBClient
- **ScanToEmail:** Scanner $\rightsquigarrow$ ScanIP $\rightsquigarrow$ IP1 $\rightsquigarrow$ IP2 $\rightsquigarrow$ USBClient
- **ProcessFromStore:** USBClient $\rightsquigarrow$ IP1 $\rightsquigarrow$ IP2 $\rightsquigarrow$ USBClient
- **PrintWithProcessing:** USBClient $\rightsquigarrow$ IP2 $\rightsquigarrow$ PrintIP

Here $A \rightsquigarrow B$ means that the start of processing by A should precede the start of processing by B. In the *DirectCopy* datapath, a job is processed in order by the components Scanner, ScanIP, IP1, and IP2, and then simultaneously sent to the Controller via the USBClient and to the printer through PrintIP. The interpretation of the remaining datapaths is similar.

It is not mandatory that components in a datapath process a job sequentially: the design of the system allows for a certain degree of parallelism. Scanner and ScanIP, for instance, may process a page in parallel. This is because ScanIP processes the output of the scanner on a line-by-line basis ("streaming") and has the same throughput as the Scanner. However, due to the characteristics of the different components, some additional constraints are imposed. Because of the nature of the image processing function that IP2 performs, IP2 can start processing a page only after IP1 has completed processing it. The dependency between ScanIP and IP1 is different. IP1 processes the output of ScanIP in streaming mode and has a higher throughput than ScanIP. Hence IP1 may start processing the page while ScanIP is processing it, with a certain delay due to the higher throughput of IP1.

In addition to the image processing components, two other system resources are memory and USB

bandwidth. Execution of a job is only allowed if the entire memory required for completion of the job is available (and allocated) before its execution commences. Each component requires a certain amount of memory for its task and this can be released once computation has finished and no other component needs the information. Another critical resource is the USB. This bus has limited bandwidth and serves as a bridge between the USBClient and the memory. The bus may be used both for uploading and for downloading data. At most one job may upload data at any point in time, and similarly at most one job may download data. Uploading and downloading may take place concurrently. We assume that the transmission rate for both concurrent and non-concurrent bus usage is constant.

2.2   Model

In the Uppaal model, each use case and each resource is described by an automaton, except for memory which is simply modeled as a shared variable.

All image processing components, as well as the USB, follow the same behavioral pattern, displayed in Fig. 2. Initially a component is in idle mode. As soon as the component is claimed by a job, it enters the running mode. An integer variable execution_time specifies how long the automaton stays in this mode. After this period has elapsed, the automaton jumps to the recovery mode, and stays there for recover_time time units. The template of Fig. 2 is parameterized by channel names start_resource and end_resource, which mark the start and termination of a component, and integer variables execution_time and recover_time, which describe the timing behavior.
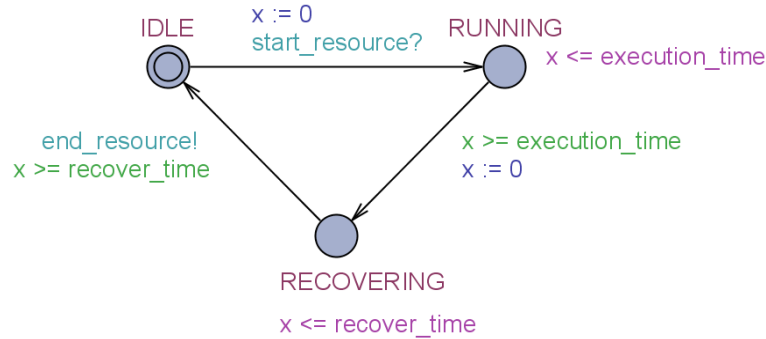

Figure 2: Component template

Each use case is modeled using a separate automaton. As an example, the automaton for the *DirectCopy* use case is depicted in Fig. 3. A job may only claim the first component from its datapath when enough memory is available for all the processing in the datapath. This figure shows the way memory allocation and release is modeled. At the moment a component is claimed, the use case automaton specifies its execution time. For reasons of space we have not included the definition of functions like setScannerTime() in this paper; for the full definitions we refer to the Uppaal model available at `http://www.mbsd.cs.ru.nl/publications/papers/fvaan/TigaOce/`. The figure illustrates, also, the way we modeled the parallel activities of IP2, USBClient and PrintIP. After finishing with the last component in the datapath, the automaton loops back to the INIT location so that a new job can start. This loop models a queue of *DirectCopy* jobs.

The *Observer* automaton, depicted in Fig. 4, is used to measure the throughput of jobs. Every time a *DirectCopy* job finishes, it synchronizes with an *Observer* instance, via the observe channel, to reset the
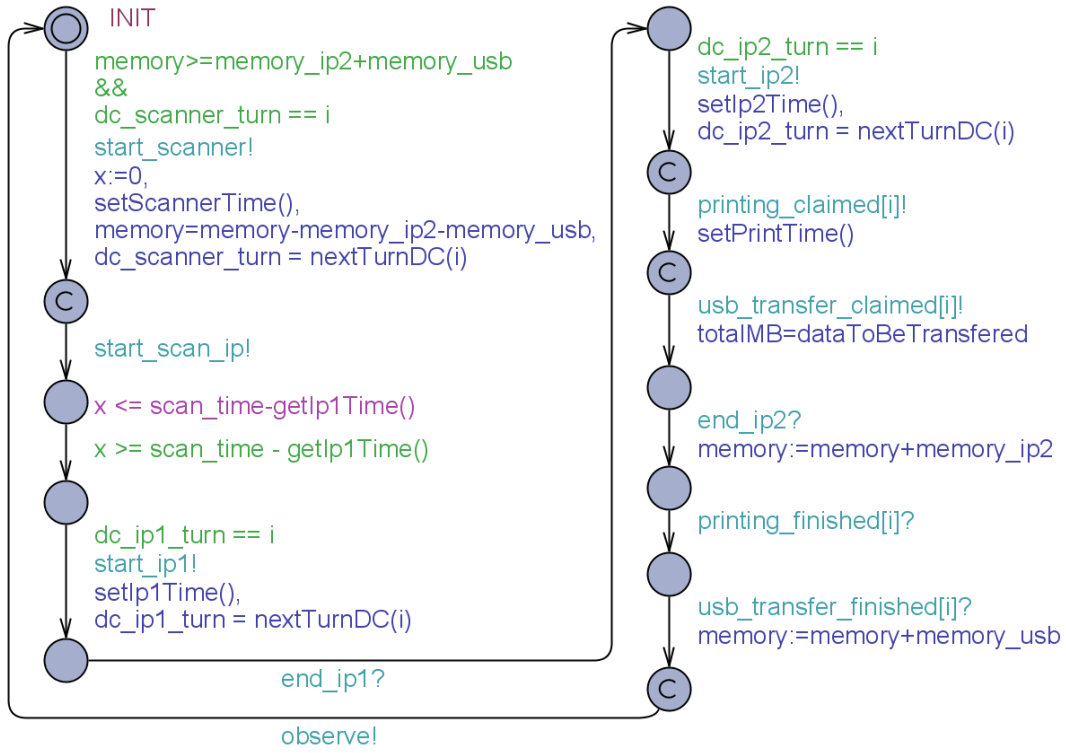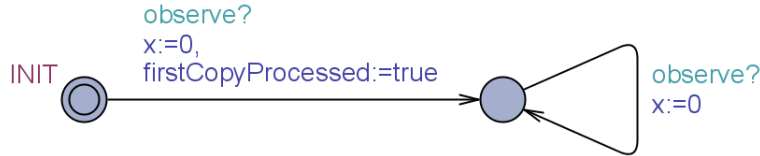
INIT

memory>=memory_ip2+memory_usb
&&
dc_scanner_turn == i
start_scanner!
x:=0,
setScannerTime(),
memory=memory-memory_ip2-memory_usb,
dc_scanner_turn = nextTurnDC(i)

start_scan_ip!

x <= scan_time-getIp1Time()
x >= scan_time - getIp1Time()

dc_ip1_turn == i
start_ip1!
setIp1Time(),
dc_ip1_turn = nextTurnDC(i)

end_ip1?

observe!

dc_ip2_turn == i
start_ip2!
setIp2Time(),
dc_ip2_turn = nextTurnDC(i)

printing_claimed[i]!
setPrintTime()

usb_transfer_claimed[i]!
totalMB=dataToBeTransfered

end_ip2?
memory:=memory+memory_ip2

printing_finished[i]?

usb_transfer_finished[i]?
memory:=memory+memory_usb

Figure 3: *DirectCopy* template

observe?
x:=0,
firstCopyProcessed:=true

INIT

observe?
x:=0

Figure 4: Observer template

observer clock x. The *Observer* automaton has two locations because the first job being processed can have a different throughput requirement than the subsequent jobs. When the first job has been processed the flag firstCopyProcessed is set to true. Section 3 describes how winning conditions can be expressed using *Observer* instances.

In order to reduce the state space, our model also incorporates a few of the scheduling rules that are commonly used in practice. However, we only included rules that do not rule out optimal schedules. A first scheduling rule that we included is *non-overtaking*, which states that different instances of the same use case may not overtake each other. As described in [4], non-overtaking reduces the state space without losing optimal schedules. Another rule that we included in our model is *non-laziness* [1]. Here the idea is that whenever a job needs a resource and this resource is available, the resource is allocated immediately, either to this job or to another one. In general, non-laziness may rule out optimal schedules, but we only introduce non-laziness for resources for which there is no competition. In a setting where there is no competition for memory, this reduces the state space without losing optimal schedules. Nonlaziness can

be implemented in Uppaal by making the channels for claiming a resource urgent.

3   Uppaal Tiga and Model Adaptation

In this section, we first describe a realistic scenario that illustrates the importance of unpredictable jobs, that is, jobs with uncertain arrival times. Then we investigate how Uppaal and Uppaal Tiga can be used to model these unpredictable jobs. Based on the results of this investigation we present an adaptation of the model described in Section 2.

3.1   A Realistic Scenario

The system described in Section 2 has several different datapaths that use the system components in different ways. In practice, a common scenario is that the printer/copier processes a series of *DirectCopy* jobs and at some time a *PrintWithProcessing* job arrives. One may think of a person making a direct copy of a document and someone else starting a print job on the same machine from a remote location, at a time that is not known in advance. The *PrintWithProcessing* job should be served in a reasonable time. However the *DirectCopy* jobs should not be delayed for too long. Therefore we would like to find a strategy that can deal with the unpredictable nature of the *PrintWithProcessing* job and still ensure an acceptable trade-off between the throughput of the *PrintWithProcessing* and the *DirectCopy* jobs.

3.2   Unpredictable Jobs in Uppaal

Uncertainty of job arrival times can be modeled in Uppaal using non-determinism. However, when we ask Uppaal to compute an optimal schedule, Uppaal will chose the arrival times of jobs in such a way that this allows for the optimal schedule. Since the Uppaal tool is based on timed automata, rather than timed game automata, we cannot introduce a distinction between controllable and uncontrollable delays.

A simple scenario that demonstrates this problem is described here. Suppose we have two *PrintWith-Processing* jobs where one has a predictable arrival time and the other job is unpredictable in terms of arrival time. The trade-off property specifies that when the unpredictable job arrives it has a very low serve time compared to the predictable job. Both jobs need access to the *USBclient*, which is the first component in the data path of both jobs. This makes the *USBclient* the critical resource. If the unpredictable job needs a critical resource, Uppaal will ensure that this critical resource is not claimed by the predictable job in the optimal schedule. The reason for this is that the low serve time of the unpredictable job has to be satisfied quickly while the predictable job can be postpone without violating its serve time. Thus the strategy behind the optimal schedule has some knowledge about the *future* arrival time of an unpredictable job in order to ensure that the critical resource is not claimed by a predictable job.

3.3   Unpredictable Jobs in Uppaal Tiga

As described earlier, Uppaal Tiga defines a two player game with on the one side the controller (in charge of the controllable edges) and on the other side the environment (in charge of the uncontrollable edges). This strict separation between controller and environment can be used to generate strategies that do not depend on any knowledge of the arrival time of unpredictable jobs. Intuitively we can see Uppaal as having one player, the controller, that has control over all edges. In Uppaal Tiga we can use the concept of a game to separate control into two players. The idea is that the player representing the environment has full control over the arrival time of unpredictable jobs. The goal of the environment

is to prevent the controller from satisfying the control objective by activating the unpredictable jobs at the worst possible time. In the simple scenario described in the previous paragraph this would be at the time when the predictable job has claimed the critical *USBclient* resource. The controller has to find a strategy to satisfy the trade-off constraint despite the efforts of the environment. This prevents the controller strategy from having any knowledge about the arrival time of the unpredictable jobs. Thus Uppaal Tiga can be used to investigate scenarios, like the one described in Section 3.1, that deal with unpredictable jobs.

3.4 Model Adaptation

In order to reflect the unpredictable arrival time of a *PrintWithProcessing* job, the first transition of the corresponding automaton is made uncontrollable, as shown in Figure 5. This automaton represents a looping *PrintWithProcessing* by having the last transition pointing back to the INIT location. The clock TimeSinceArrival measures the serve time of the current *PrintWithProcessing* job and is reset upon arrival of a new job. According to the observer automaton discussed in Section 2.2, the guard firstCopyProcessed of the first transition ensures that the *PrintWithProcessing* job arrives after the printer has processed the first *DirectCopy* job.
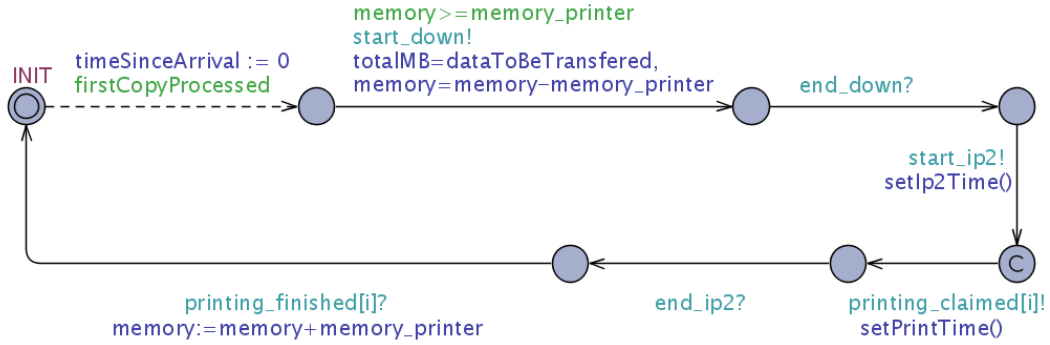

Figure 5: The automaton of *PrintWithProcessing* job in Tiga

With our adaptation of the model, it is easy to specify the winning condition of the strategy being generated. We use two looping instances of *DirectCopy* jobs (DC) and one looping instance of a *PrintWithProcessing* job (DP). The actual number of processed jobs is unbounded, but we make the assumption that there are only two DC and one DP job at the same time. If there is only one concurrent DP job, two instances of DC jobs are enough to keep the scanner busy all the time. The winning condition can be expressed as follows:

```
control:A[] (DC_OBSERVER.INIT imply DC_OBSERVER.x <= FIRST_DC_TIME)  &&
(!DC_OBSERVER.INIT imply DC_OBSERVER.x <= DC_TIME)  &&
(!DP0.INIT imply DP0.timeSinceArrival <= DP_TIME)
```

As described in Section 2.2, the first line states that the first DC job has to be served within FIRST_DC_TIME time units. We can chose an arbitrary value here as long as the first job can be processed within that time,

without changing the result. The only thing that is expressed here is that the first job eventually finishes. After the first DC job has been served, the DP jobs may arrive at unpredictable times. The variables `DC_TIME` and `DP_TIME` together represent a trade-off between the throughput of DC and the serve time of DP jobs. The second and third line express that such a trade-off should be satisfied.

We tried different combinations of values of the serve time of DCs (`DC_TIME`) and the serve time of DP (`DP_TIME`) to find out the trade-off where a strategy exists. The idea is, for a fixed value of the serve time of one kind of job, to determine the minimal value of the serve time for the other kind of job.

4   Results

In this section, we report on the results obtained with Uppaal Tiga for the Océ case study.

4.1   Optimal Strategies

We found six optimal strategies, each providing a different trade-off between the throughput of *Direct-Copies* and the maximal serve time of *PrintWithProcessing* jobs. These optimal strategies are represented as a *Pareto frontier* in Figure 6. Note that, since constants `DC_TIME` and `DP_TIME` may only take integer
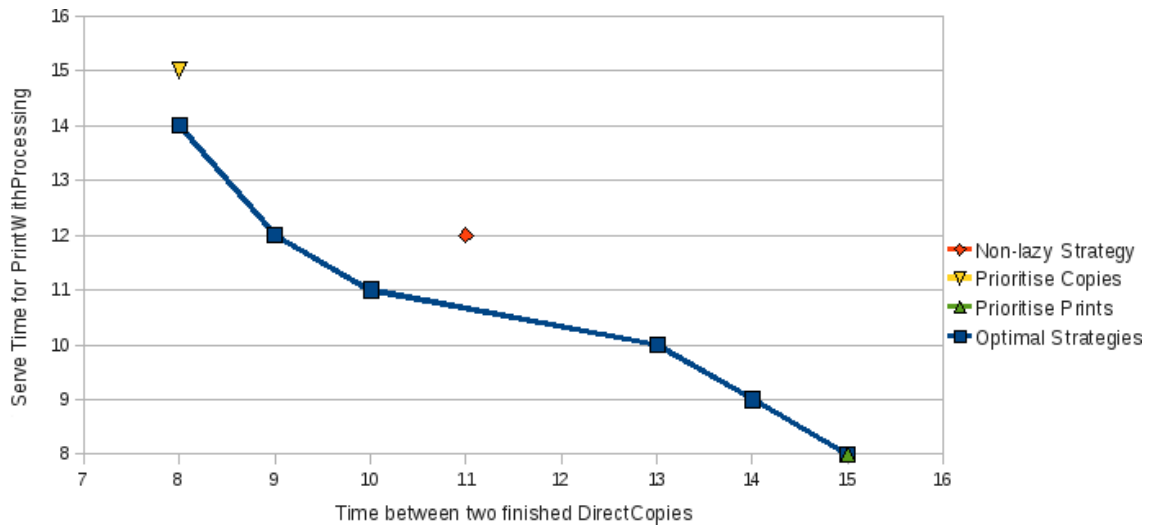

Figure 6: Pareto Frontier Representing Optimal Strategies & Timings for Fixed Strategies

values, no formal meaning can be attached to points on the frontier except those with integer coordinates. The minimal serve time for *PrintWithProcessing* jobs is 7 in the case where no other jobs are present. However, there is no strategy achieving this under the condition that the *PrintWithProcessing* jobs should finish eventually. For *PrintWithProcessing* a strategy for achieving the lowest possible serve time can be found. This strategy leads to a throughput for *DirectCopies* with is more than twice as low as the optimal one.

There are some gaps in the data-points. For example, for a maximal time between two finished *DirectCopy* jobs of 11 and 12 there is no corresponding time on the y-axis. This is because the values for both are 11, which is the same as with a lower maximal time between two finished *DirectCopy* jobs of 10. So these strategies are always worse and therefore we did not depict them in the diagram.

Uppaal Tiga also offers the possibility to export the generated strategies as a list of rules. However, the strategies created by Tiga are not suitable to be built directly into a real printer controller. The first reason for this is that the set of rules is huge and contains thousands of rules. The second reason is that the rules contain information about the state of all automata, including for instance the auxiliary agent automata which are not present in a real system. Also, the strategy only works for a situation with a fixed number of jobs and is therefore not generally applicable.

4.2   Simple Strategies

Although the strategies generated by Uppaal Tiga cannot directly be used within a real printer controller, they can still be helpful for improving the scheduling of a printer. In practice, the scheduling strategy of a printer should be very simple. What we can do is to implement simple strategies in our model and compare the timings for them with the optimal ones found by Uppaal Tiga. In Figure 6 three data-points representing the performance of fixed strategies are given.

The first very simple strategy is to make all resources non-lazy (that is, all channels urgent). In theory this does not entirely fix the strategy because if two jobs are waiting for the same resource there is a nondeterministic choice which jobs gets the resource at the moment it becomes available. However for the small number of jobs in our scenario this situation is not relevant and we get a fixed strategy. As one can see, the timings for this simple strategy are worse than the optimal ones.

We also considered two strategies that prioritize one kind of job. For prioritising *DirectCopies* we only allow a *PrintWithProcessing* job to use IP2 when the scanner is not in use. With this strategy the throughput for the *DirectCopies* is optimal but the serve time of the *PrintWithProcessing* jobs is higher than with the optimal strategy. This is because the scanner uses more time than IP2 and consequently there are situations where using IP2 is not a problem because it will be free again at the moment the scanner is finished.

In the strategy that gives priority to *PrintWithProcessing* jobs, a *DirectCopy* may not use IP2 if a *PrintWithProcessing* job arrives at the USB bus. But this is not sufficient, since there is more than one instance of *DirectCopy*. It may happen that a *DirectCopy* is still using the printer when the *PrintWith-Processing* job finishes using IP2 and wants to use the printer. To solve this issue, we added another constraint that a *DirectCopy* may only use IP2 if another instance of *DirectCopy* is not using the printer. While this decreases the average throughput, for the situation where we want to prioritise another kind of job, never two *DirectCopies* are printed without another kind of job between them. So we get a strategy which is as good as the optimal one for this case.

4.3   Results for an Extended Model

We have also experimented with an extended version of the model presented in Section 2. This extended model includes many features of an Océ printer which is currently in the design phase[1]. One example of such a feature is the memory bus, which is the interface between the components and the memory. Memory management was also refined, for instance, print jobs access different memory locations than scan jobs. Besides these, the numerical values were similar to those used in reality.

The most important change w.r.t. the model of Section 2 is the modeling of the scheduling rules employed by the printer's controller. One example of such a rule is a memory bus arbitration, which is based on a priority mechanism between the components. Another example is the rule used for releasing

---

[1]Because of space limit we cannot add a full description of the model, but we invite the reader to have a look at the model available at `http://www.mbsd.cs.ru.nl/publications/papers/fvaan/TigaOce/`.

the print job memory when the Print and the USB Upload actions are delayed. With these rules, the model essentially becomes fully deterministic. However, in Uppaal (Tiga) no partial order reduction mechanism is implemented. As a result of large numbers of (confluent) synchronization transitions, we could not analyze scenarios with a large number of jobs and big amount of memory available. We mention here one example of a scenario which we could analyze. We have used two different datapaths: ScanToEmail and PrintWithProcessing where Print is added in parallel with USBClient. In this scenario two looping ScanToEmail jobs are present in the system together with one looping PrintWithProcessing job. Non-lazy scheduling is used, memory can buffer one print job and one scan job and 10 Scan to Email jobs are processed. For this scenario, Uppaal Tiga could compute the fastest schedule (strategy) when one job type is prioritized. The only simplification we made in this model is that we reduced the memory size so that the system can store fewer jobs. If we remove some of the scheduling rules so that the model becomes (essentially) nondeterministic again, Uppaal Tiga is no longer able to handle the model.

5   Conclusions & Future Work

In this paper, we have applied Uppaal Tiga to automatically compute adaptive scheduling strategies for an industrial case study dealing with a state-of-the-art image processing pipeline of a printer. As far as we know, this is the first application of timed automata technology to an industrial scheduling problem with uncertainty in job arrivals.

Despite our promising initial results, the problem to automatically synthesize *practical* scheduling strategies for this application domain is still wide open. In some other applications, the strategies synthesized by Tiga have been used directly in the generation of control software [11, 8]. This is currently not possible for the printer case study described in this paper: the strategies produced by Tiga (albeit memoryless) are really big and contains tens of thousands of rules. Even though there appears to be room to improve Tiga so that it generates smaller strategies, direct use of Tiga strategies for scheduling of printers will probably not be practical for the years to come. We had to restrict our analysis to a scenario with a fixed continuous stream of direct copy jobs and a single uncontrollable direct print job, simply because Tiga cannot handle more uncontrollable jobs. Also, the model that we described in Section 2 is a simplification of the realistic and more complex models that we constructed in the context of the Octopus project. For realistic models, the generated strategies will even be much bigger and far beyond the tight constraints on CPU and memory usage of todays printer controllers.

Nevertheless, we believe that we are close to the point at which Uppaal Tiga can already be useful in the actual design of data path controllers. These controllers typically consists of a relatively small number of simple rules that determine which resource is allocated to which job (for instance: job and resource priorities, FCFS for jobs with same priority, greedy resource allocation,..). Due to state space explosion, Tiga will not be able to handle unconstrained realistic printer models. However, our experiments show that Tiga is able to deal with realistic printer models of which the state space is constrained by a few of the simple scheduling rules that Océ wants to use anyway. By applying Tiga to (downsized) versions of printer models and replaying the strategies in the simulator, we may be able to come up with simple new control rules that are implementable on real printers. Tiga may also give an indication of how close the implemented rules are from the optimum.

References

[1] Y. Abdeddaïm, E. Asarin & O. Maler (2006): *Scheduling with timed automata. Theor. Comput. Sci.* 354(2), pp. 272–300.

[2] R. Alur & D.L. Dill (1994): *A theory of timed automata. Theoretical Computer Science* 126, pp. 183–235.

[3] AMETIST (2007). *Final Project Report.* Available at `http://www.cs.ru.nl/F.Vaandrager/AMETIST/final.pdf`. Deliverable from the European project IST-2001-35304 Advanced Methods for Timed Systems (AMETIST).

[4] G. Behrmann, E. Brinksma, M. Hendriks & A. Mader (2005): *Production Scheduling by Reachability Analysis - A Case Study.* In: *19th International Parallel and Distributed Processing Symposium (IPDPS 2005), CD-ROM / Abstracts Proceedings, 4-8 April 2005, Denver, CA, USA.* IEEE Computer Society. Available at `http://doi.ieeecomputersociety.org/10.1109/IPDPS.2005.363`.

[5] G. Behrmann, A. Cougnard, A. David, E. Fleury, K.G. Larsen & D. Lime (2007). *Uppaal Tiga User-manual.* Available at `http://www.cs.aau.dk/~adavid/tiga/manual.pdf`.

[6] G. Behrmann, A. David & K.G. Larsen (2004): *A Tutorial on Uppaal.* In: M. Bernardo & F. Corradini, editors: *Formal Methods for the Design of Real-Time Systems, International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM-RT 2004, Bertinoro, Italy, September 13-18, 2004, Revised Lectures, Lecture Notes in Computer Science* 3185. Springer, pp. 200–236.

[7] F. Cassez, A. David, E. Fleury, K.G. Larsen & D. Lime (2005): *Efficient On-the-Fly Algorithms for the Analysis of Timed Games.* In: Martín Abadi & Luca de Alfaro, editors: *CONCUR 2005 - Concurrency Theory, 16th International Conference, CONCUR 2005, San Francisco, CA, USA, August 23-26, 2005, Proceedings, Lecture Notes in Computer Science* 3653. Springer, pp. 66–80. Available at `http://dx.doi.org/10.1007/11539452_9`.

[8] F. Cassez, J.J. Jessen, K. G. Larsen, J.-F. Raskin & P.-A. Reynier (2009): *Automatic Synthesis of Robust and Optimal Controllers - An Industrial Case Study.* In: Rupak Majumdar & Paulo Tabuada, editors: *Hybrid Systems: Computation and Control, 12th International Conference, HSCC 2009, San Francisco, CA, USA, April 13-15, 2009. Proceedings, Lecture Notes in Computer Science* 5469. Springer, pp. 90–104. Available at `http://dx.doi.org/10.1007/978-3-642-00602-9_7`.

[9] A.J. Davenport & J.C. Beck (2000). *A Survey of Techniques for Scheduling with Uncertainty.* Unpublished manuscript.

[10] G. Igna, V. Kannan, Y. Yang, T. Basten, M. Geilen, F. Vaandrager, M. Voorhoeve, S. de Smet & L. Somers (2008): *Formal Modeling and Scheduling of Datapaths of Digital Document Printers.* In: *Proceedings Sixth International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2008),* September 15-17, 2008, Saint-Malo, France, *Lecture Notes in Computer Science* 5215. Springer Berlin / Heidelberg, pp. 169–186. Available at `http://www.cs.ru.nl/ita/publications/papers/fvaan/Octopus08.html`.

[11] J.J. Jessen, J. Rasmussen, K.G. Larsen & A. David (2007): *Guided Controller Synthesis for Climate Controller Using Uppaal Tiga.* In: Jean-François Raskin & P. S. Thiagarajan, editors: *Formal Modeling and Analysis of Timed Systems, 5th International Conference, FORMATS 2007, Salzburg, Austria, October 3-5, 2007, Proceedings, Lecture Notes in Computer Science* 4763. Springer, pp. 227–240.

[12] O. Maler, A. Pnueli & J. Sifakis (1995): *On the Synthesis of Discrete Controllers for Timed Systems.* In: E.W. Mayr & C. Puech, editors: *Proceedings STACS'95, Lecture Notes in Computer Science* 900. Springer-Verlag, pp. 229–242.

[13] K.N. McKay, J.A. Buzacott & F.R. Safayeni (1989): *The Scheduler's Knowledge of Uncertainty: The Missing Link.* In: J. Browne, editor: *Knowledge Based Production Management Systems.* Elsevier Science, New York, pp. 171–189.

[14] K.N. McKay, F.R. Safayeni & J.A. Buzacott (1998): *Job-shop scheduling theory: What is relevant? Interfaces* 18(4), pp. 84–90.

[15] K.N. McKay & V.C.S. Wiers (1999): *Unifying the Theory and Practice of Production Scheduling. Journal of Manufacturing Systems* 18(4), pp. 241–255.